A Parallel and Distributed Framework for Constraint Solving
(abstract)

Vasco Pedro[1], Rui Machado[2], and Salvador Abreu[1]

[1] CENTRIA and Departamento de Informática,
Universidade de Évora, Portugal – {vp,spa}@di.uevora.pt
[2] Franhofer ITWM,
Kaiserslautern, Germany – rui.machado@itwm.fhg.de

Abstract. With the increased availability of affordable parallel and distributed hardware, programming models for these architectures has become the focus of significant attention. Constraint programming, which can be seen as the encoding of processes as a Constraint Satisfaction Problem, because of its data-driven and control-insensitive approach is a prime candidate to serve as the basis for a framework which effectively exploits parallel architectures.
To effectually apply the power of distributed computational systems, there must be an effective sharing of the work involved in the search for a solution to a Constraint Satisfaction Problem (CSP) between all the participating agents, and it must happen dynamically, as it is hard to predict the effort associated with the exploration of some part of the search space.
We describe and provide an initial experimental assessment of an implementation of a work stealing-based approach to distributed CSP solving, which relies on multiple back-ends for the distributed computing mechanisms – from the multicore CPU to supercomputer clusters running MPI or other interprocess communication platforms.

1 Introduction

Constraints are used to model problems with no known polynomial algorithm, but for which search techniques developed within the field of constraint programming provide viable procedures.

Notwithstanding their relative efficiency, constraint solving methods are computationally demanding and good candidates to benefit from multiprocessing. Moreover, the declarative style of constraint programming frees the programmer from concerns usually entailed by parallel and distributed programming, such as control, synchronisation, and communication issues. In fact, the programmer may not even be aware that there is any parallelism involved in solving the problem.

Given the increasing availability of parallel computational resources, in the form of multiprocessors, clusters of computers, or both, there is a need for an

effective way to help incorporating that power into the constraint programming setting. In this context, our goal is to build a library which takes advantage of parallel hardware in a transparent way, for constraint solving.

In parallel constraint solving (see for example [2, 4, 1]) the problem may be partitioned around the domains of the variables, effectively partitioning its search space. The search for a solution is then carried out in each of the sub-search spaces by one agent (or worker), all agents working in parallel.

Constraint solving involves exploring large search spaces. To perform search using several agents in parallel, the effort ought to be shared among them. This may happen either by having each agent do a part of the work and coordinate with the other agents, or the agents may be mostly independent from each other, performing their (possibly non-overlapping) part of the work, hoping that one of them finds a quicker path to an answer. While the former typically requires significant inter-agent communication, not only for the search to progress but also for termination detection, in the latter communication can be limited to an initial dispatching of the agents and to an answer collecting phase at the end of the procedure. In that case, however, the initial work distribution may turn out to be quite unbalanced, leaving some agents to bear most of the effort while others become idle and their contribution is wasted.

This article reports on preliminary results of our experiments in implementing PaCCS, a work-stealing scheme for overcoming the effect described above. This is a two-level scheme: work stealing occurs between co-located agents, but when distant agents are involved, some cooperation is needed to redistribute the work still left.

Moreover, as we aim to seek high performance on hierarchical hybrid multiprocessors (such as clusters of multicore computers), the underlying interprocess communication facility is determining with respect to the resulting performance: we have to adapt to whatever programming patterns are adequate to yield performance. Nevertheless, this must be done without burdening the application programmer.

2 The PaCCS Solver Architecture

A PaCCS constraint solver consists of *workers*, grouped together as *teams* (Figure 1). The search for one or all solutions is carried out by the workers, which implement a propagator-based constraint solving engine. Each active worker has a pool of *idle* search spaces and a *current* search space, the one it is currently working on. Each team includes a *controller*, which does not participate in the search. One of the controllers, the *main controller*, also coordinates the teams.

Structuring the workers this way serves two purposes: the first is that a workers' sole task becomes searching, as all communication with the environment required by the dynamic sharing of work among teams is handled by the controller. The second objective is the sharing of resources enabled by binding the workers in a team close together. If all workers were on the same level, they would either have to divide their attention between search and communication

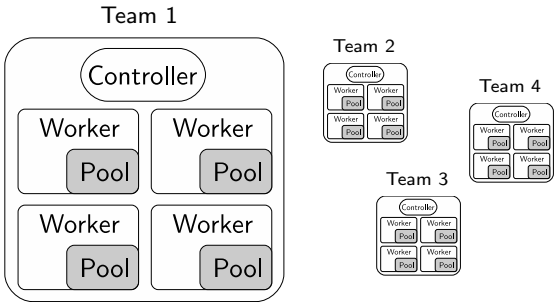

Fig. 1. PaCCS Solver Architecture

or there would have to be one controller per worker, thereby increasing resource usage.

3 Distributed Computational Models

Our initial implementation of PaCCS is based on POSIX threads and MPI. We forked PaCCS to derive MaCS, which is based on a Partitioned Global Address Space (PGAS) interprocess communications framework called GPI, developed at Fraunhofer ITWM, and for which our initial experiments on unbalanced tree search [3] have yielded good performance indicators.

4 Conclusions and Future Work

We do not provide formal experimental results, as the prototype implementation is still undergoing intense development. We do however, note that we carried out experiments with up to 256 cores on 64 nodes, with interesting speedups.

The present prototype implementation of PaCCS is written in C and is designed to allow for experimentation with several parameters. This goal has been met and the system can be tuned to work on several multiprocessor organizations. The work leading up to this can be partly found in [5] and [3]. We are presently working on tuning PaCCS and MaCS to be able to tackle large problems and generally improve performance.

References

1. Geoffrey Chu, Christian Schulte, and Peter J. Stuckey. Confidence-Based Work Stealing in Parallel Constraint Programming. In Ian P. Gent, editor, *CP*, volume 5732 of *Lecture Notes in Computer Science*, pages 226–241. Springer, 2009.
2. Pascal Van Hentenryck. Parallel Constraint Satisfaction in Logic Programming: Preliminary Results of CHIP within PEPSys. In *ICLP*, pages 165–180, 1989.
3. Rui Machado, Carsten Lojewski, Salvador Abreu, and Franz-Josef Pfreundt. Unbalanced tree search on a manycore system using the GPI programming model. *Computer Science - R&D*, 26(3-4):229–236, 2011.

4. Laurent Michel, Andrew See, and Pascal Van Hentenryck. Parallelizing Constraint Programs Transparently. In Christian Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 514–528. Springer, 2007.
5. Vasco Pedro and Salvador Abreu. Distributed work stealing for constraint solving. *CoRR - Proceedings of CICLOPS-WLPE 2010*, abs/1009.3800, 2010.